In Python, the \_\_new\_\_ method plays a crucial role in object creation. Let’s dive into the details:

1. **Purpose of \_\_new\_\_**:

* When you create an instance of a class, Python first calls the \_\_new\_\_() method to **create the object** and then calls the \_\_init\_\_() method to **initialize the object’s attributes**.
* The \_\_new\_\_() method is a **static method** of the object class.
* Its signature is: object.\_\_new\_\_(class, \*args, \*\*kwargs).
* The first argument is the **class** of the new object you want to create, followed by optional \*args and \*\*kwargs.

1. **Creating a New Object**:

* The \_\_new\_\_() method should return a new instance of the class.
* You can override \_\_new\_\_ to perform custom actions before and after creating an instance.
* To create an object of a class, you typically call super().\_\_new\_\_(...).

1. **Example**:

class Person:   
def \_\_init\_\_(self, name):   
 self.name = name   
   
person = Person('John') 

* When you create an instance using person = Person('John'), Python internally calls both \_\_new\_\_() and \_\_init\_\_() methods.
* Equivalent method calls:
* person = object.\_\_new\_\_(Person, 'John')
* person.\_\_init\_\_('John')
* The \_\_dict\_\_ of the person object:
* After \_\_new\_\_(): {} (empty)
* After \_\_init\_\_(): {'name': 'John'}

1. **Method Sequence**:

* When creating an object by calling the class:

class Person:   
def \_\_new\_\_(cls, name):   
 print(f'Creating a new {cls.\_\_name\_\_}object...')   
 obj = object.\_\_new\_\_(cls)   
return obj   
   
def \_\_init\_\_(self, name):   
 print(f'Initializing the person object...')   
 self.name = name   
   
person = Person('John') 

* Output:

Creating a new Person object...

Initializing the person object... 

Remember, \_\_new\_\_ is responsible for object creation, while \_\_init\_\_ initializes the object’s attributes. The order of execution is \_\_new\_\_ first, followed by \_\_init\_\_. 🌟🐍